AMENDMENTS TO THE CLAIMS

This listing of the claims replaces all prior versions and listings of claims in this application.

Listing of the Claims:

1. (Previously Presented) An embedded processor, comprising

A. a plurality of processing units, each executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads"),

B. one or more execution units that are shared by, and in communication coupling with, the plurality of processing units, the execution units executing instructions from the threads,

C. an event delivery mechanism that delivers events to respective threads with which those events are associated, wherein the event delivery mechanism

i. is in communication coupling with the plurality of processing units, and

ii. delivers each such event to the respective thread without execution of instructions by said processing units.

2. (Original) The embedded processor of claim 1, wherein the thread to which an event is delivered processes that event without execution of instructions outside that thread.

3. (Original) The embedded processor of claim 1, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

4. (Original) The embedded processor of claim 1, wherein the execution units execute instructions from the threads without need to know what thread they are from.

5. (Original) The embedded processor of claim 1, wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread.

6. (Original) The embedded processor of claim 1, wherein at least one of the processing units is a virtual processing unit.

7. (Previously Presented) An embedded processor, comprising

A. a plurality of virtual processing units, each executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads"), wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread,

B. a plurality of execution units,

C. a pipeline control that is in communication coupling with the plurality of processing units and with the plurality of execution units, the pipeline control launching instructions from plural ones of the threads for concurrent execution on plural ones of the execution units,

D. an event delivery mechanism that is in communication coupling with the plurality of processing units and that delivers events to respective threads with which those events are associated without execution of instructions by said processing units, where the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events;,

E. wherein a thread to which such an event is delivered processes that event without execution of instructions outside that thread.

8. (Original) The embedded processor of claim 7, where the pipeline control comprises a plurality of instruction queues, each associated with a respective virtual processing unit.

9. (Original) The embedded processor of claim 8, where the pipeline control decodes instruction classes from the instruction queues.

10. (Original) The embedded processor of claim 8, where the pipeline control controls access by the processing units to a resource providing source and destination registers for the instructions dispatched from the instruction queues.

11. (Original) The embedded processor of claim 8, wherein the execution units include a branch execution unit responsible for any of instruction address generation, address translation and instruction fetching.

12. (Original) The embedded processor of claim 11, wherein the branch execution unit maintains state for the virtual processing units.

13. (Original) The embedded processor of claim 7, where the pipeline control controls access by the virtual processing units to the execution units.

14. (Original) The embedded processor of claim 7, where the pipeline control signals a branch execution unit that is shared by the virtual processing unit as the instruction queue for each virtual processing unit is emptied.

15. (Original) The embedded processor of claim 7, where the pipeline control idles the execution units to decrease power consumption.

16. (Original) The embedded processor of claim 7, wherein the plurality of execution units include any of integer, floating, branch, compare and memory units.

17. (Previously Presented) An embedded processor system, comprising

A. a plurality of embedded processors,

B. a plurality of virtual processing units executing on the plurality of embedded processors, each virtual processing executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") and each thread being any of constrained or not constrained to execute on a same virtual processing unit and/or a same processor during a life of that thread,

C. one or more execution units that are shared by, and in communication coupling with, the plurality of virtual processing units, the execution units executing instructions from the threads, the execution units including any of integer, floating, branch, compare and memory execution units:,

D. an event delivery mechanism that delivers events to respective threads with which those events are associated, wherein the event delivery mechanism

i. is in communication coupling with the plurality of virtual processing units, and

ii. delivers each such event to the respective thread without execution of instructions by said virtual processing units.

18. (Original) The embedded processor system of claim 17, wherein the thread to which an event is delivered processes that event without execution of instructions outside that thread.

19. (Original) The embedded processor system of claim 18, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

20. (Original) The embedded processor system of claim 18, wherein the branch unit is responsible for fetching instructions that are to be executed for the threads.

21. (Original) The embedded processor system of claim 20, wherein the branch unit is additionally responsible for any of instruction address generation and address translation.

22. (Original) The embedded processor system of claim 21, wherein the branch unit comprises thread state stores that store thread state for each of the respective virtual processing units.

23. (Original) The embedded processor system of claim 18, comprising

A. a pipeline control that is in communication coupling with the plurality of processing units and with the plurality of execution units, the pipeline control dispatching instructions from plural ones of the threads for concurrent execution on plural ones of the execution units,

B. the pipeline control comprises a plurality of instruction queues, each associated with a respective virtual processing unit, and wherein

C. instructions fetched by the branch execution unit are placed in the instruction queues associated with the respective virtual processing unit in which the corresponding thread is executed.

24. (Original) The embedded processor system of claim 23, wherein one or more instructions are fetched at a time for a said thread with a goal of keeping the instructions queues at equal levels.

25. (Original) The embedded processor system of claim 24, wherein the pipeline control dispatches one or more instructions at a time from a given instruction queue for execution.

26. (Original) The embedded processor system of claim 25, wherein a number of instructions dispatched by the pipeline control at a given time from a given instruction queue is controlled by a stop flag in a sequence of instructions in that queue.

27. (Original) The embedded processor system of claim 23, wherein the pipeline control launches, and the execution units execute, multiple instructions from one or more threads simultaneously.

28. (Previously Presented) An embedded processor, comprising

A. a plurality of processing units, each executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads"), wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread,

B. an event delivery mechanism that delivers events to respective threads with which those events are associated, wherein the event delivery mechanism

i. is in communication coupling with the plurality of processing units, and

ii. delivers each such event to the respective thread without execution of instructions by said processing units.

29. (Original) The embedded processor of claim 28, wherein the thread to which an event is delivered processes that event without execution of instructions outside that thread.

30. (Original) The embedded processor of claim 28, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

31. (Original) The embedded processor of claim 28, wherein at least one of the processing units is a virtual processing unit.

32. (Previously Presented) An embedded processor system, comprising

A. a plurality of embedded processors,

B. a plurality of virtual processing units executing on the plurality of embedded processors, each virtual processing executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") and each virtual processing unit being any of constrained or not constrained to execute on a same virtual processing unit and/or a same processor during a life of that thread,

C. an event delivery mechanism that delivers events to respective threads with which those events are associated, wherein the event delivery mechanism

i. is in communication coupling with the plurality of virtual processing units, and

ii. delivers each such event to the respective thread without execution of instructions by said virtual processing units.

33. (Original) The embedded processor system of claim 32, wherein the thread to which an event is delivered processes that event without execution of instructions outside that thread.

34. (Original) The embedded processor system of claim 33, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

35. (Previously Presented) A method of embedded processing, comprising the steps of

A. executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") on each of a plurality of processing units,

B. executing instructions from the threads in one or more execution units that are shared by the plurality of processing units,

C. delivering events to respective threads with which those events are associated without executing instructions by said processing units.

36. (Currently Amended) The method of claim 35, comprising the step of processing the event by the thread to which the ~~thread~~ event is delivered without execution of instructions outside that thread.

37. (Original) The method of claim 35, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

38. (Original) The method of claim 35, wherein the step of executing instructions from the threads in one or more execution units does not necessitate the execution units knowing what threads the respective instructions are from.

39. (Original) The method of claim 35, wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread.

40. (Original) The method of claim 35, wherein at least one of the processing units is a virtual processing unit.

41. (Previously Presented) A method of embedded processing, comprising the steps of

A. executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") on each of a plurality of virtual processing units, wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread;,

B. launching instructions from plural ones of the threads for concurrent execution on a plurality of execution units,

C. delivering events to respective threads with which those events are associated without execution of instructions by said virtual processing units, where the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events, and

D. processing the event to which the thread is delivered without execution of instructions outside that thread.

42. (Original) The method of claim 41, where the launching step includes decoding instruction classes from the instruction queues.

43. (Original) The method of claim 41, where the launching step includes controlling access by the virtual processing units to a resource providing source and destination registers for the instructions dispatched from the instruction queues.

44. (Original) The method of claim 41, comprising executing any of the steps of instruction address generation, address translation and instruction fetching using a branch execution unit that is shared by all of the virtual processing units.

45. (Original) The method of claim 44, comprising maintaining state for the virtual processing units using the branch execution unit.

46. (Original) The method of claim 41, wherein the execution units include any of integer, floating, branch, compare and memory units.

47. (Previously Presented) A method of embedded processing, comprising the steps of

A. executing a plurality of virtual processing units on a plurality of embedded processors,

B. executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") on each of a plurality of virtual processing units, each virtual processing unit being any of constrained or not constrained to execute on a same virtual processing unit and/or a same embedded processor during a life of that thread,

C. executing instructions from the threads in one or more execution units that are shared by the plurality of virtual processing units, the execution units including any of integer, floating, branch, compare and memory execution units, and

D. delivering events to respective threads with which those events are associated without execution of instructions by said virtual processing units.

48. (Currently Amended) The method of claim 47, comprising processing the event by the thread to which the ~~thread~~ event is delivered without execution of instructions outside that thread.

49. (Original) The method of claim 48, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

50. (Original) The method of claim 48, comprising fetching instructions that are to be executed for the threads using the branch execution unit.

51. (Original) The method of claim 50, comprising executing with the branch execution unit any of instruction address generation and address translation.

52. (Original) The method of claim 48, comprising

A. dispatching instructions from plural ones of the threads for concurrent execution on plural ones of the execution units,

B. the dispatching step including placing instructions fetched for each of the respective threads in instruction queues associated with the virtual processing units in which that thread is executed.

53. (Original) The method of claim 52, wherein the dispatching step includes fetching one or more instructions at a time for a given thread with a goal of keeping the instructions queues at equal levels.

54. (Original) The method of claim 53, wherein the dispatching step includes dispatching one or more instructions at a time from a given instruction queue for execution by the execution units.

55. (Original) The method of claim 54, wherein a number of instructions dispatched by the pipeline control at a given time from a given instruction queue is controlled by a stop flag in a sequence of instructions in that queue.

56. (Original) The method of claim 52, comprising launching and executing multiple instructions from one or more threads simultaneously.

57. (Previously Presented) A method of embedded processing, comprising the steps of

A. executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") on each of a plurality of processing units, wherein each thread is any of constrained or not constrained to execute on a same processing unit during a life of that thread,

B. delivering events to respective threads with which those events are associated without execution of instructions by said processing units.

58. (Original) The method of claim 57, comprising the step of processing the event to which the thread is delivered without execution of instructions outside that thread.

59. (Original) The method of claim 57, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.

60. (Original) The method of claim 57, wherein at least one of the processing units is a virtual processing unit.

61. (Previously Presented) A method of embedded processing, comprising the steps of

A. executing a plurality of virtual processing units on a plurality of embedded processors,

B. executing one or more processes or threads (which one or more processes or threads are collectively referred to as "threads") on each of a plurality of virtual processing units, each thread being any of constrained or not constrained to execute on a same virtual processing unit and/or a same embedded processor during a life of that thread, and

C. delivering events to respective threads with which those events are associated without execution of instructions by said virtual processing units.

62. (Currently Amended) The method of claim 61, wherein processing the event by the thread to which the ~~thread~~ event is delivered without execution of instructions outside that thread.

63. (Original) The method of claim 62, wherein the events include any of hardware interrupts, software-initiated signaling events ("software events") and memory events.